Bio 264 Homework #3

Nicholas J. Gotelli

Due: 22 February 2016

Create a new markdown file called `<Your last name>_HW#3_22Feb2016`. Cut and paste the problems below into Markdown, then add chunks of R code to provide your answers. You will turn in both the `.Rmd` and the `.html` file for this homework assignment.

1. Create a function that takes as input an integer value n. Inside your function, create a square n x n matrix and fill it with zeroes. Now replace the diagonal elements with 1s. The function should print the matrix. If you have done this correctly, the 1s should form a visual pattern of an "x" that extends to the 4 corners of the square matrix.

- Try to do this exercise using a `for` loop.

The key to this problem is to notice that, for any square n x n matrix with row i and column j, we can create an "x" pattern by 1) first filling all of the cases in which i==j and 2) adding a 1 whenever the sum of the row and column coefficients equals the number of rows + 1. We want both conditions, so we need to setup an if statement with the OR operator | to catch both in 1 statement

```
if (i + j== n + 1 | i==j) m[i,j] <- 1
```

Now we just insert this into our double `for` loop. To me, this seems much more elegant than the solution below, which requires several steps.

```r
MyDiag1 <- function(n=5){
  m <- matrix(rep(0,n^2),nrow=n)
  for (i in 1:n){
    for (j in 1:n){
      if(i + j == n + 1 | i==j) m[i,j] <- 1
    }
  }
  return(m)
}
MyDiag1()
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    1
## [2,]    0    1    0    1    0
## [3,]    0    0    1    0    0
## [4,]    0    1    0    1    0
## [5,]    1    0    0    0    1
```

```r
MyDiag1(8)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1    0    0    0    0    0    0    1
## [2,]    0    1    0    0    0    0    1    0
```

```
## [3,]    0    0    1    0    0    1    0    0
## [4,]    0    0    0    1    1    0    0    0
## [5,]    0    0    0    1    1    0    0    0
## [6,]    0    0    1    0    0    1    0    0
## [7,]    0    1    0    0    0    0    1    0
## [8,]    1    0    0    0    0    0    0    1
```

- Hunt around on line or in the help system and seem if you can use some of R's matrix functions to achieve the same thing.

Here is one way to do this. We first use the **`diagonal`** *function to fill in all the diagonal elements with 1s. Next, we create a matrix z that has the row and column subscripts for the elements with a 1. Next we adjust the z matrix to get the "mirror" column on the other side of the diagonal. Finally, we change those elements in the z matrix from 0 to 1. I have added intermediate print statements so you can see the output at each step, but you would normally strip those out once the code was debugged. This solution works, but I find it a bit cumbersome.*

```
MyDiag <- function(n=5){
  m <- matrix(rep(0,n^2),nrow=n) # create a square matrix of zeroes
  print(m)                       # print matrix
  diag(m) <- 1                   # fill diagonal elements with 1s
  print(m)                       # print matrix
  z <- which(m==1,arr.ind=TRUE)  # get coefficients for the 1s
  print(z)                       # print coefficient list
  z[,2] <- n + 1 - z[,2]         # set "mirror" element for the columns
  print(z)                       # print modified coefficient list
  m[z] <- 1                      # change the z elements
  return(m)
}

MyDiag()
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    0
## [2,]    0    0    0    0    0
## [3,]    0    0    0    0    0
## [4,]    0    0    0    0    0
## [5,]    0    0    0    0    0
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    0
## [2,]    0    1    0    0    0
## [3,]    0    0    1    0    0
## [4,]    0    0    0    1    0
## [5,]    0    0    0    0    1
##      row col
## [1,]   1   1
## [2,]   2   2
## [3,]   3   3
## [4,]   4   4
## [5,]   5   5
##      row col
## [1,]   1   5
## [2,]   2   4
```

```
## [3,]    3    3
## [4,]    4    2
## [5,]    5    1
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    1
## [2,]    0    1    0    1    0
## [3,]    0    0    1    0    0
## [4,]    0    1    0    1    0
## [5,]    1    0    0    0    1
```

2. The most basic equation of all is a straight line:

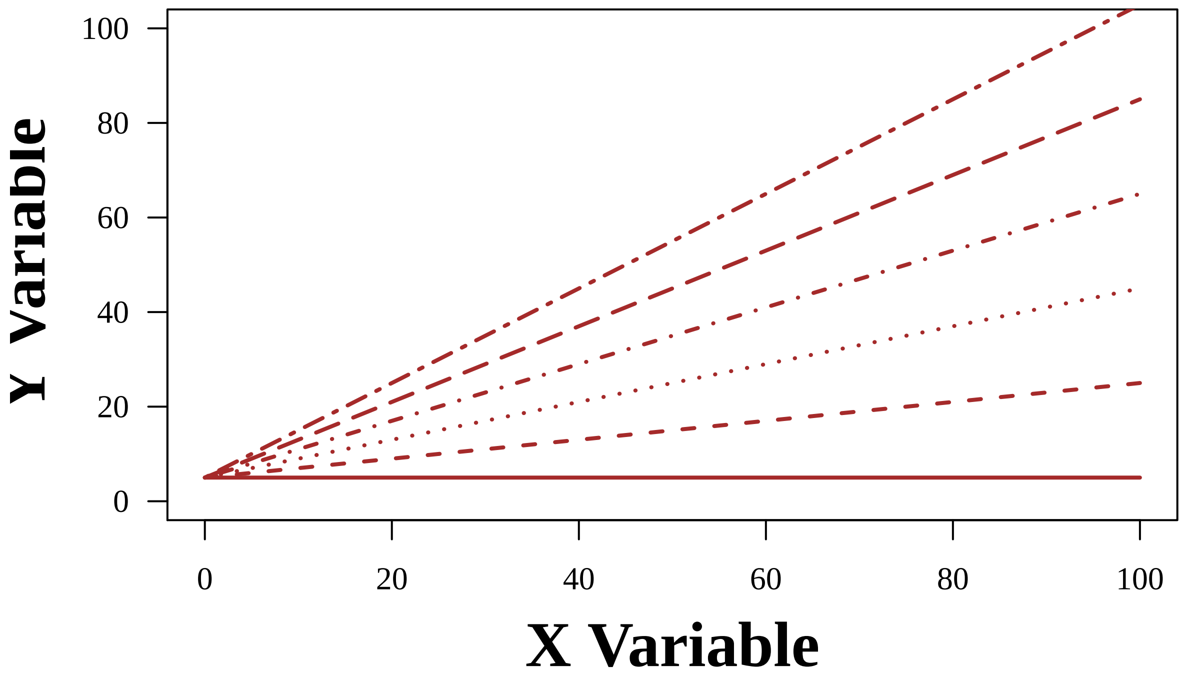
$$Y = a + bX$$

where Y is the "dependent variable", X is the "independent variable", and a and b are the constants.

- Using the methods we have developed in class, create a graphic function for examining this linear relationship and illustrate it with different settings for the two parameters. What is the interpretation of the parameters a and b
- Use this as an opportunity to explore the many options that are available in `par`, and `plot` (or `matplot`). Spiff up your margins, labels, and graphs so they look sharp!

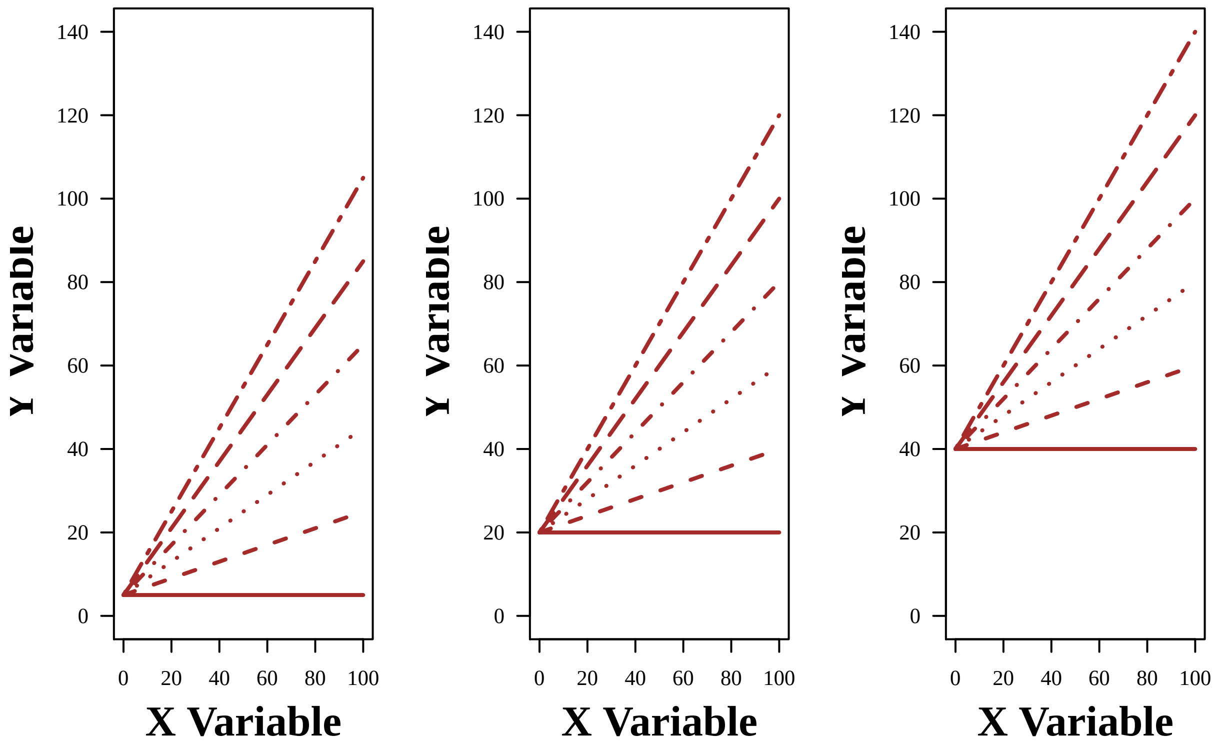
Here is a simple function for varying the slope, with a few of the many possible parameters illustrated for `mplot`. I then use it to create a 3 panel graph in which I vary the intercept:

```
MyLine <- function(a=5,b=seq(0,1,length=6),x=0:100,ylim=c(0,100)){
  m <-matrix(nrow=length(x),ncol=length(b))
 for (i in 1:length(b)){
 y <- a + b[i]*x
 m[,i] <- y
 }




 matplot(x=x,y=m,
      lty=1:length(b),     # uses different line types
      lwd=2,               # increases line thickness
      type="l",            # lines with no points
      pty="s",             # square plotting space
      xlab="X Variable",   #x label
      ylab="Y Variable",   #y label
      ylim=ylim,           #user defined ylimits (adjust for multiple plots)
      las=1,               # plot axis numbers all horizontal
      col="brown",         # change line colors
      cex.lab=2,           # increase label size
      family="serif",      # use serif font for labels
      font.lab=2)          # use boldface for labels
}
MyLine()
```



And here are 3 panels illustrating variation in both the slope and the intercept:

```
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3))
a <- c(5,20,40) # set different choices for the intercept
for (i in 1:3){
  MyLine(a=a[i],ylim=c(0,140))
}
```


140
120
100
80
60
40
20
0
Y Variable
0 20 40 60 80 100
X Variable
140
120
100
80
60
40
20
0
Y Variable
0 20 40 60 80 100
X Variable
140
120
100
80
60
40
20
0
Y Variable
0 20 40 60 80 100
X Variable

```
par(opar)
```